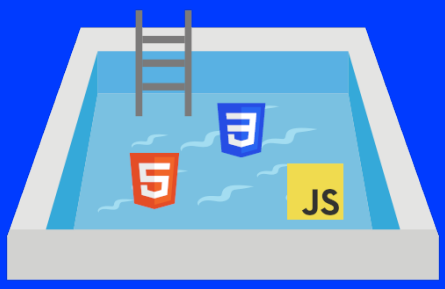
{EPITECH}

WEB DEV POOL_

< DAY 10 - AJAX />

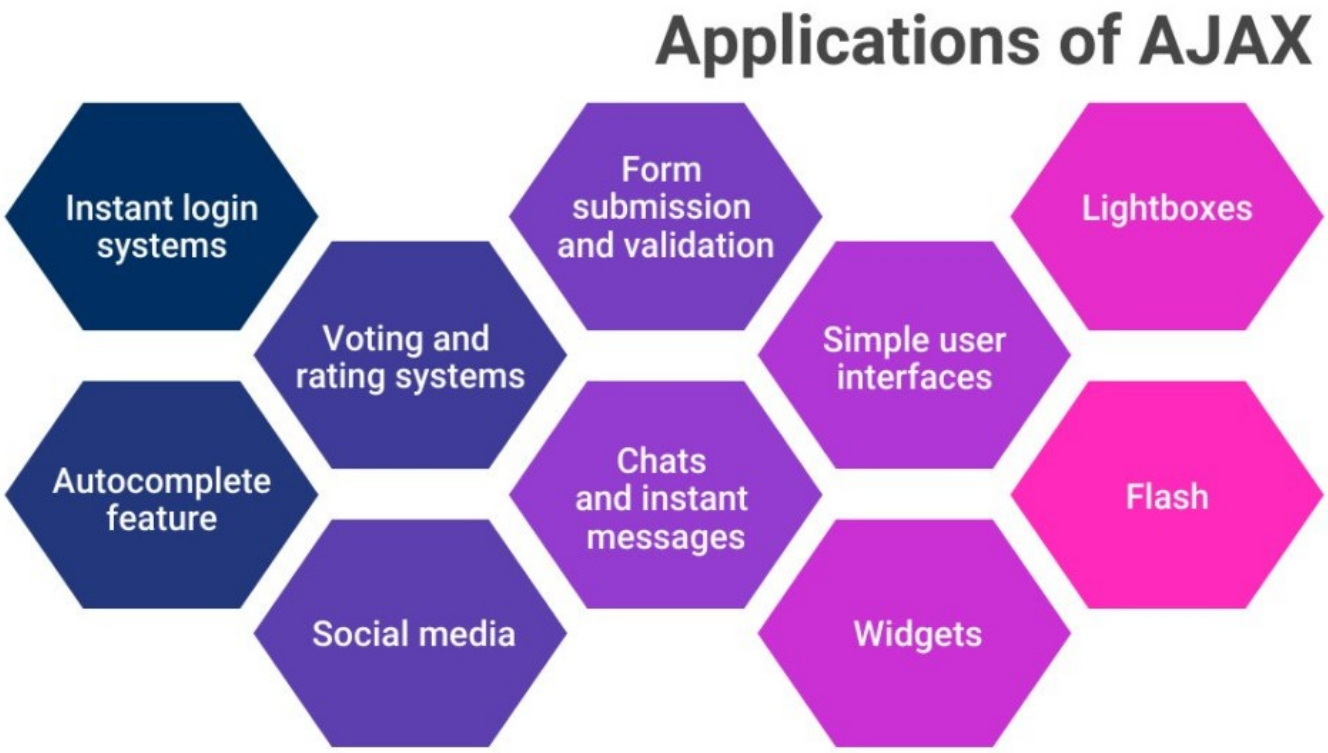


WEB DEV POOL

With **parallelization**, you can break a large sequential task into smaller pieces that can be processed independently and simultaneously. Pretty cool, right?

You can achieve the outcomes much faster, your system gets easier to evolve and is more resilient. Parallel computing requires **asynchronous programming**.

Asynchronous **JA**vascript and **X**ML enables communication between a client (your web browser) and a server (apache/nginx) without refreshing your web page.



Asynchronous and callbacks

So far, you always programmed is in a synchronous way. It means that each statement is carried out in the order you specified. The *asynchronous* way uses **callbacks**.

It is a function, called only when a particular event is triggered.

For instance: a request to a slow/small server takes time. If it is synchronous, the user must wait for the request to complete before doing anything else.
With asynchronous programming, the user can continue doing what they want while the request is executed.

Some precisions

We will use the ”old school” way of doing things for server side code.
You'll be asked to work with echo and PHP's JSON encoder.

You will later see in your studies that this way is totally out of the picture.
Instead of echoing JSON data, many frameworks allow you to create objects like `JsonResponse` that replace the practice that we're going to ask you to use.
Feel free to go and check these out.

First things first: create the skeleton for the html page you will be using this whole day.
You need to use the **Bootstrap** framework.

You are going to use Bootstrap to make an alert notification appear.
Feel free to check the documentation to learn how to use it.

Include the JavaScript file in your following html page: `bootstrap/js/bootstrap.min.js`.

Look at the Javasript section on getbootstrap.com and use some of the Javascript components in a new Javascript file (for instance `js/main.js`). Include it in your html page too.

There are two remaining thing to do before starting the tasks:

1. Read the MDN documentation on the **Fetch API**.
2. Read it twice... Really, I mean it!

Task 01

Delivery: `task01.html`, `task01.php`

The goal of this task is to have a small form composed of a text field for entering a name and a button which, when pressed, makes an asynchronous GET call to your PHP script.

When called, your script must `echo` a JSON object containing **name** as key and the input as value.

You MUST specify in the response header that you are going to communicate JSON data.

Your Javascript code MUST be in script tags. It's a very short script, so you don't need a file for it.

Upon success you will create an alert notification displaying the name given by the PHP script.
In case of an error, display that an error occured.

Task 02

Delivery: `task02.html`, `task02.php`

In your HTML, add a form containing an input ”email”.
When the form is submitted, it POSTs a request to the server containing the serialized data.
In your PHP script, check the email address validity.

In case of an error or if the email address is not valid, change the HTTP status code to 400.

You must display an alert notification indicating whether the email address is valid or not.

Task 03

Delivery: `task03.html`

Resources: `countries.json`

Let's make use of the provided file `countries.json`.

In your HTML page, create:

- ✓ an empty table with 2 columns (**country-code** and **name**) ;
- ✓ a **load countries** button, that fills the table with the data contained in the file.

In order to make this feature you MUST use the **json()** method of the `Response` interface.

Starting from task 04, you must have a MySQL or MariaDB server setup.

Task 04

Delivery: `task04.html`, `task04.php`, `task04.js`

Create a database and import data using the provided `ajax_products.sql`.
Then, create an HTML page with 2 fields that sends 2 strings.
The goal is to display if a brand can be added in the database previously created.

You must always display two validation messages in real-time, one for each field, dynamically updating with each modification.

Requirements:

- ✓ **type** allows alphabetical characters and `-`, iIts length is between 3 and 10 included ;
- ✓ **brand** allows alphanumeric characters, `-` and `&`, its length is between 2 and 20 included ;
- ✓ both fields are case insensitive ;
- ✓ a GET request to your server is made using `fetch()`, in a `task04.js` file ;
- ✓ your server must then contact your database to fulfill the request.

Validation messages:

- ✓ $type: this type does not have enough characters.
- ✓ $type: this type has too many characters.
- ✓ $type: this type has non-alphabetical characters (different from '-').
- ✓ $type: this type doesn't exist in our shop.
- ✓ $type: this type exists in databases.
- ✓ No type sent yet!
- ✓ $brand: this brand does not have enough characters.
- ✓ $brand: this brand has too many characters.
- ✓ $brand: this brand has invalid characters.
- ✓ $brand: this brand already exists in databases.
- ✓ $brand: this brand is valid for the type $type.
- ✓ No brand sent yet!

These messages MUST be highlighted in red for errors and green for valid inputs.

Task 05

Delivery: `task05.html`, `task05.php`, `task05.js`

Starting from the previous exercise, add 2 fields: `price` and `number`.
The goal is to perform a search and display the result in an array.

Requirements:

- ✓ **price** allows only digits and >, < or =, its length is between 2 and 5 (e.g.: '>100', '<42' or '=42') ;
- ✓ **number** field only allows positive numbers ;
- ✓ these two new fields are also case insensitive ;
- ✓ a GET request to your server is made using `fetch()`, in a `task05.js` file ;
- ✓ the response MUST contain JSON, not HTML ;
- ✓ your server must then contact your database to fulfill the request ;
- ✓ a valid request let you display, without refreshing the HTML page, the products in a 5-column-array (type, brand, price, number, stock) ;
- ✓ the content of the array must be cleared when a new search is done.

After clicking the button, display an error message if the request does NOT succeed.

Error messages:

- ✓ Error ($type): this type does not have enough characters.
- ✓ Error ($type): this type has too many characters.
- ✓ Error ($type): this type has non-alphabetical characters (different from '-').
- ✓ Error ($type): this type doesn't exist in our shop.
- ✓ Error ($brand): this brand does not have enough characters.
- ✓ Error ($brand): this brand has too much characters.
- ✓ Error ($brand): this brand has invalid characters.
- ✓ Error ($brand): this brand doesn't exist in our database.
- ✓ Error ($price): this price does not have enough characters.
- ✓ Error ($price): this price has too many characters.
- ✓ Error ($price): we cannot define a price - string invalid.
- ✓ Error ($price): no products found at this price.
- ✓ Error ($number): sorry, we don't have enough stock, we only have $stock in stock.
- ✓ Error ($number): not a positive number.

Task 06

Delivery: `task06/*`

Create a mini-chat with asynchronous programming:

- ✓ it contains 2 text boxes: one for the name, and another one for the message ;
- ✓ it displays the older messages and the sender's name.

⚠️ Messages must be refreshed every 4 seconds.

📢 Be creative, you are free to use everything you've just seen.

v 4.1